PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-066904
(43)Date of publication of application : 03.03.2000

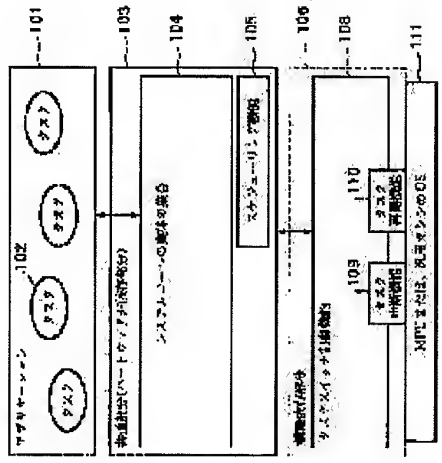
(51)Int.Cl. G06F 9/46

(21)Application number : 10-236031
(22)Date of filing : 21.08.1998

(71)Applicant : CANON INC
(72)Inventor : HIRAHARA ATSUSHI
BESSHO MASATAKA
IWABUCHI YOICHI
YAMADA JUNJI

(54) METHOD FOR CONTROLLING MULTITASK AND STORAGE MEDIUM

(57)Abstract:
PROBLEM TO BE SOLVED: To enable application to execute the same operation by having the same interface specification on all platforms in multitask control for controlling an application program including plural tasks.
SOLUTION: A common part 103 executes scheduling related to the execution of plural application tasks 102 included in application 101 and issues a task control instruction such as task interruption processing and task restarting processing to a machine kind depending part 106. The depending part 106 controls a task based on the task control instruction issued from the common part 103. In this case, when the operation environment of the application 101 is on an MPU in a real machine, a task interruption function 109 and a restart function 110 directly rewrite the contents of a task control block. On the other hand, when the operation environment is on a general OS, the task is controlled by using a system call supported by the general OS.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号
特開2000－66904
（P2000－66904A）
(43)公開日　平成12年３月３日(2000.3.3)

(51)Int.Cl.'	識別記号	ＦＩ		テーマコード'(参考)
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｂ	5Ｂ０９８

審査請求　未請求　請求項の数11　ＯＬ　（全 17 頁）

(21)出願番号　特願平10－236031

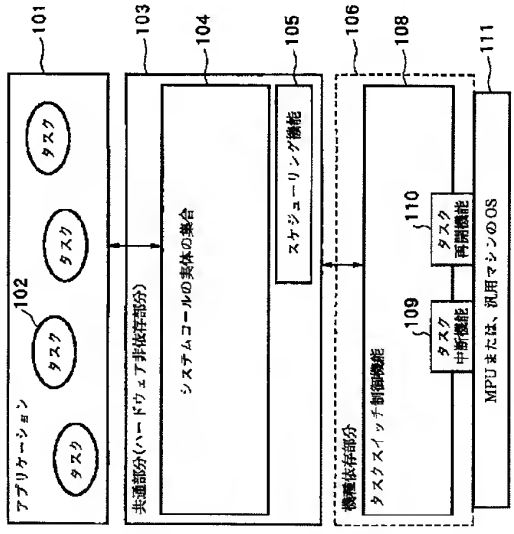
(22)出願日　平成10年８月21日(1998.8.21)

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号
(72)発明者　平原　厚志
東京都大田区下丸子３丁目30番２号　キヤノン株式会社内
(72)発明者　別所　正隆
東京都大田区下丸子３丁目30番２号　キヤノン株式会社内
(74)代理人　100076428
弁理士　大塚　康徳　（外２名）

最終頁に続く

(54)【発明の名称】　マルチタスク制御方法及び記憶媒体

(57)【要約】
【課題】複数のタスクを含むアプリケーションプログラムを制御するためのマルチタスク制御において、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことを可能とする。
【解決手段】共通部分１０３は、アプリケーション１０１に含まれる複数のアプリケーションタスク１０２の実行に関わるスケジューリングを行い、タスク中断処理やタスク再開処理等のタスク制御指示を機種依存部分１０６に発行する。機種依存部分１０６は、共通部分１０３より発行されたタスク制御指示に基づいてタスクの制御を行う。ここで、当該アプリケーションの動作環境が実機のＭＰＵ上であれば、タスク中断機能１０９、再開機能１１０はタスクコントロールブロックの内容を直接書き換える。一方、動作環境が汎用ＯＳ上であれば、当該汎用ＯＳがサポートするシステムコールを用いてタスクの制御を行う。



1

【特許請求の範囲】

【請求項１】　複数のタスクを含むアプリケーションプログラムを制御するためのマルチタスク制御方法であって、

前記複数のタスクの実行に関わるスケジューリングを行い、タスク制御指示を発行するスケジューリング工程と、

前記スケジューリング工程で発行されたタスク制御指示に基づいて、当該アプリケーションプログラムの動作環境に応じてタスクの実行を制御する実行工程とを備えることを特徴とするマルチタスク制御方法。

【請求項２】　前記タスク制御指示はタスクの中断指示及び再開指示を含み、

前記実行工程は、前記スケジューリング工程で発行された中断指示及び再開指示に基づいて、当該アプリケーションプログラムの動作環境に応じてタスクの中断処理及び再開処理を実行することを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項３】　前記実行工程は、前記タスク制御指示に基づいてタスクを制御するためのタスクコントロールブロックのデータを書き換えることによりタスクを制御することを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項４】　前記実行工程は、前記タスク制御指示に基づいて汎用ＯＳのシステムコールを発行することによりタスクを制御することを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項５】　前記スケジューリング工程におけるタスク制御指示がタスクの生成指示及び削除指示を含むことを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項６】　前記実行工程は、更に、前記アプリケーションプログラムの動作環境に応じた割込み処理が可能であることを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項７】　前記実行工程は、汎用ＯＳからの割込み信号に応じて割込み処理を実行するためのスレッドを生成し、割込み処理を実行することを特徴とする請求項６に記載のマルチタスク制御方法。

【請求項８】　前記スケジューリング工程で発行したタスク制御指示について履歴情報を保持する保持工程と、前記保持工程で保持された履歴情報を前記アプリケーションプログラムの動作環境に応じた形態で出力する出力工程とを更に備えることを特徴とする請求項１に記載のマルチタスク制御方法。

【請求項９】　コンピュータに、複数のタスクを含むアプリケーションプログラムを制御させるためのマルチタスク制御プログラムを格納する記憶媒体であって、該マルチタスク制御プログラムが、

前記複数のタスクの実行に関わるスケジューリングを行い、タスク制御指示を発行するスケジューリング工程のコードと、

2

前記スケジューリング工程で発行されたタスク制御指示に基づいて、当該アプリケーションプログラムの動作環境に応じてタスクの実行を制御する実行工程のコードとを備えることを特徴とする記憶媒体。

【請求項１０】　前記マルチタスク制御プログラムが、前記スケジューリング工程で発行したタスク制御指示について履歴情報を保持する保持工程のコードと、

前記保持工程で保持された履歴情報を前記アプリケーションプログラムの動作環境に応じた形態で出力する出力工程のコードとを更に備えることを特徴とする請求項９に記載の記憶媒体。

【請求項１１】　前記スケジューリング工程のコードがＣ言語プログラムのテキストで記述されていることを特徴とする請求項９に記載の記憶媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、マルチタスク処理が可能なマルチタスク制御方法及び記憶媒体に関する。

【０００２】

【従来の技術】一般に、リアルタイムＯＳ（以下、ＲＴＯＳという）では、実機上で動作するＲＴＯＳをワークステーションやＰＣなどの汎用開発マシン上で模擬実行するためのＲＴＯＳシミュレーション開発環境が存在する。

【０００３】これらのＲＴＯＳシミュレーション環境には、（１）ＲＴＯＳのシステムコールのスタブを用意し、アプリケーション・タスク毎の単体テストを開発マシン上で行うもの、（２）ターゲットのＭＰＵの動作を模擬するシミュレータを用いて、ターゲット用のクロスコンパイルで生成したＲＴＯＳとアプリケーションを動作させるもの、（３）ターゲット用ＲＴＯＳのハードウェア依存部分を改変して、それを汎用マシンのＯＳ（ＵＮＩＸやＷｉｎｄｏｗｓなど）が提供するマルチスレッド機構等を利用し、アプリケーション・タスクをスレッドとして実現するもの、などが存在する。

【０００４】特に（２），（３）の方法では、実機上のＲＴＯＳとほぼ同じ関数インターフェースを持つように作られている。これにより、実機上とほぼ同じアプリケーション・プログラムをＲＴＯＳシミュレータ上で動作させることが可能である。

【０００５】また、ＲＴＯＳアプリケーションのデバッグ環境として、実機上ではＩＣＥや、ＲＯＭエミュレータを介して、もしくはリモートデバッグなどの手法により、開発マシンと実機を接続し、開発マシン上で動作するソースレベルデバッガを利用するものが一般的である。また、開発マシン上で動作させるＲＴＯＳシミュレータに対するデバッグ環境としては、上記（２）の場合においては、通常、ＭＰＵのシミュレータが機械語もしくはＣ言語などのソースレベルのデバッガを備えてい

る。また、上記（１），（３）に関しては、開発マシン上で動作させるネイティブのソースレベルデバッガを使用してデバッグ／テストを行うことができる。

【０００６】ところが、ＲＴＯＳアプリケーションのデバッグにおいては、これらのソースレベルデバッガを用いてブレークをかけてアプリケーションの動作を一時停止させたり、ステップ実行を行うデバッグスタイルが必ずしも効果的に行えるとは限らない。つまり、リアルタイム性が求められるこの種のアプリケーションにおいては、プログラムを停止させた瞬間にそのアプリケーションの再現性が失われてしまうからである。

【０００７】そのため、近年ではアプリケーションプログラムを停止させること無く、システムコールやタスクスイッチの履歴を記録しておき、しかる後にその情報を取り出して開発マシン上などでグラフィカルに表示し、アプリケーションタスクの動きが意図したとおりであるかを解析することができるツール（以降、タスクスイッチ・ビューアと呼ぶ）が開発され普及し始めている。

【０００８】また、特開平５－２８２１６０の「リアルタイム・シミュレーション開発機構」のようにダイナミック・ローディング方式を採用し、実機とシミュレーション開発環境でアプリケーションをまったく変更せずに動作させることを目的としたシステムも存在する。

【０００９】

【発明が解決しようとする課題】しかしながら、上記（１）の方法では、タスク単体の振る舞いの確認を行うのみであり、タスク間の同期・通信に関わるテストを行うことはできない。また、上記（２）の方法には、実行速度が遅いために、大きな規模のアプリケーションのテストにおいてリアルタイム性がないという問題や、効率上の問題が存在する。これに対して上記（３）の方法は、シミュレーション開発環境の実現が比較的容易で、実行速度の面でも実機に最も近い振る舞いをするとされている。しかしながら、（３）の方法においても、アプリケーションの振る舞いやデバッグ環境に関して以下のような問題点が存在する。

【００１０】①プラットフォーム間のアプリケーションの動作が異なる：従来のＲＴＯＳでは、実機上で動作するＲＴＯＳと開発用コンピュータの汎用ＯＳ上で動作させるＲＴＯＳシミュレータの間には、同一のアプリケーションが厳密に同一の振舞を行うことが保証されていない。それは、シミュレーション環境では、汎用ＯＳ（ＵＮＩＸやＷｉｎｄｏｗｓなど）が提供するマルチスレッド機構をそのまま利用してタスク（スレッド）間の同期・通信機能などを実現しているためである。すなわち、汎用ＯＳが提供するマルチスレッド機構では、その汎用ＯＳ上でスケジューリングされるため、同時に動く他のプロセスや周辺デバイスへのアクセスの影響等で、スケジューリングの結果が変化する可能性がある。

【００１１】②プラットフォーム間の移植の問題：同様の問題で、ＲＴＯＳシミュレータを動作させる汎用ＯＳを変更する場合（例えば、ＵＮＩＸからＷｉｎｄｏｗｓへ移植する等）、それぞれの汎用ＯＳで提供しているマルチスレッド機構のインターフェース仕様や動作が異なるため、ＯＳシミュレータ同士の間でもアプリケーションの動作の再現性を確保するのは困難である。また、シミュレータプログラム中に、汎用ＯＳが提供するマルチスレッド機構のＡＰＩが散在しているため、ポーティング（Porting：移植）が容易ではない。

【００１２】③デバッグ情報（トレース情報）取得が困難：また、実機上でタスクスイッチ・ビューアなどのデバッグ装置を利用する場合は、実機上のＲＴＯＳのタスクスイッチを行う部分にトレース情報を取り出して記録する機能を追加し、記録された情報を何らかの方法（実機に用意されている物理的な通信手段：例えば、シリアル（＝シリアル通信；RS－232Cなどに代表されるビット単位の通信手法の一つ）やソケット（TCP／IPネットワークを利用するためのAPIの一つ）など）で外部の表示装置（タスクスイッチ・ビューアなど）に取りだす仕組みを用意すればよい。ところが、ＲＴＯＳシミュレータにおいて汎用ＯＳのスレッドの同期・通信機構を利用している場合、スレッドのコンテキストスイッチ（つまり、タスクスイッチ）はその汎用ＯＳ上で行われるため、タスクスイッチの履歴を収集するには汎用ＯＳの内部にまで立ち入る必要があり、非常に困難である。従って、ＲＴＯＳシミュレータのトレース情報を収集するのは容易ではない。

【００１３】特開平５－２８２１６０は上記①を解決することを目的としているが、汎用ＯＳで利用するシステムコールの種類の言及が無く、タスクスケジューリングはシミュレータを動作させる汎用ＯＳに依存する構成となっている。また、③に関する様な、タスクスイッチのトレース情報を表示させる仕組みは持っていない。

【００１４】本発明は上記の問題点に鑑みてなされたものであり、その目的は、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことを可能とするＲＴＯＳの構造を提供することにある。

【００１５】また、本発明の目的は、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことを可能とするＲＴＯＳ上のデバッグ環境（タスクスイッチ・ビューア等）において、あらゆるプラットフォーム上で統一的なデバッグ情報を収集することができる機構を提供することにある。

【００１６】

【課題を解決するための手段】上記の目的を達成するための本発明のマルチタスク制御方法は以下の工程を備える。すなわち、複数のタスクを含むアプリケーションプログラムを制御するためのマルチタスク制御方法であっ

て、前記複数のタスクの実行に関わるスケジューリングを行い、タスク制御指示を発行するスケジューリング工程と、前記スケジューリング工程で発行されたタスク制御指示に基づいて、当該アプリケーションプログラムの動作環境に応じてタスクの実行を制御する実行工程とを備える。

【００１７】また、本発明によれば、上記マルチタスク制御方法をコンピュータに実現させるための制御プログラムを格納する記憶媒体が提供される。

【００１８】

【発明の実施の形態】以下、添付の図面を参照して本発明の好適な実施形態を詳細に説明する。

【００１９】［第１の実施形態］図２４は第１の実施形態におけるプログラムシミュレーション環境を提供するコンピュータ装置を示す図である。１０はシミュレーション環境を提供するためのコンピュータ装置である。コンピュータ装置１０は、ＣＰＵ１１、ＲＯＭ１２、ＲＡＭ１３、入力装置１４、表示装置１５、外部記憶装置１６を備え、これらはバス１７によって接続されている。外部記憶装置１６には、実機２０において実行すべきアプリケーション１０１と、ＲＴＯＳの共通部分１０３と、ＲＴＯＳの汎用ＯＳ依存部分１０６ａが格納されている。

【００２０】一方、実機２０は制御部としてＣＰＵ２１、ＲＯＭ２２、ＲＡＭ２３、Ｉ／Ｏインターフェース２４を備え、ＲＯＭ２２に格納されたアプリケーション１０１に基づいて制御対象装置２５を制御する。ＣＰＵ２１はＲＯＭ２２に格納された制御プログラム（アプリケーション１０１、ＲＴＯＳの共通部分１０３、ＲＴＯＳのハードウエア依存部分１０６ｂ）を実行することで制御対象装置２５の制御を行うことになる。

【００２１】図１は第１の実施形態におけるＲＴＯＳの基本的な構成図である。１０１は開発対象となるアプリケーションである。１０２はアプリケーション・タスクであり、アプリケーション１０１は複数のアプリケーション・タスク１０２の集合である。個々のアプリケーション・タスク１０２はシステムコールを発行することによって、ＲＴＯＳの機能を利用する。

【００２２】１０３はハードウェアに依存しない構成要素からなるＲＴＯＳの共通部分である。１０４はＲＴＯＳシステムコールの実体の集合である。ここで用いるＲＴＯＳシステムコールは、ファイルシステムやネットワークインターフェースなどのハードウェア依存となるものは含まず、タスク制御やタスク間同期・通信機能（セマフォ、ロック、メールボックス、イベントフラグ、キュー、ｅｔｃ）など一般的なハードウェアに依存しないものを想定している。また、システムコール実行後、スケジューリング機能１０５によってタスク・スイッチのリスケジューリングが行われる。

【００２３】１０６はＲＴＯＳのハードウェア／機種に依存する構成要素群（以降、機種依存部分と呼ぶ）である。機種依存部分１０６において、１０９は現在実行中のタスクを中断する機能で、１１０は中断状態のタスクの実行を再開する機能である。１０８はタスクスイッチ制御機能で、タスク中断機能１０９およびタスク再開機能１１０を用いて、実行中のタスクを別のタスクに切り替える働きをする。なお、機種依存部分１０６は、図２４で上述した汎用ＯＳ依存部分１０６ａ、ハードウエア依存部分１０６ｂを総称したものである。１１１はＭＰＵまたは汎用ＯＳを表す。

【００２４】アプリケーション・タスク１０２などから、ＲＴＯＳに対してシステムコールの発行が行われた場合、共通部分１０３ではシステムコールの実体１０４を実行し、その結果リスケジューリングを行う必要がある場合は、スケジューリング機能１０５を呼び出してリスケジューリングを行う。タスクスイッチ制御機能１０８はタスクの中断および再開の要求を受けてタスクの切り替えを行う。

【００２５】図２は、システムコール発行からタスクスイッチまでの、共通部分による処理の手順を表したフローチャートである。システムコール関数の呼び出しを受けると、ステップＳ７０２において、タスクスイッチ制御機能１０８に対してタスク中断要求を発行する。そして、ステップＳ７０３においてシステム・コール本体１０４を実行する。本リアルタイムＯＳのシステムコール関数の内部は、機能的に２つの部分に別れている。それは、タスクの実行制御（タスクスイッチ：タスクの中断、リスケジューリング、タスクの再開制御を行なう）部分とシステムコール本体部分です。つまり、システムコールにはタスクの生成／破壊や、メッセージキュー、セマフォ、イベントフラグのペンド／ポストなどがあるが、それらの機能を実行する部分が「システム・コール」本体で、その機能を実行した結果に応じて、タスクの実行制御（タスクスイッチ）が行われる。そして、システム・コール本体１０４を実行した結果、スケジューリングの必要が生じた場合は、ステップＳ７０４からステップＳ７０５へ進み、スケジューリング機構１０５を呼び出してリスケジューリングを行う。なお、システムコールの種別や、状況によってはスケジューリングの必要がない場合がある。たとえば、新しいセマフォを生成するとか、現在のイベントフラグの値を調べる、といったシステムコールの実行はタスクスイッチの要因となる状態変化を起こさないので、その場合はスケジューリング機構を呼び出す必要は無い。逆に、セマフォ、イベントフラグのポスト・ペンドなどは、タスクスイッチの要因となる状態変化を引き起こすので、そのようなシステムコール実行後は必ずスケジューリングを行う。つまり、スケジューリングを必要とするシステムコールと必要としないシステムコールがあり、その判断はシステムコール本体でセットされるフラグ（スケジューリング要

求フラグ）により行われる。次にステップＳ７０６において、タスク制御機能１０８へタスク再開要求を発行し、本処理を終える。なお、次に実行を再開すべきタスクは、ステップＳ７０５のスケジューリング機構によって決定される。スケジューリングの必要が無い場合は、直前のタスクが選択される。従って、ステップＳ７０６では、現在選択されている次実行タスクを再開するための手続きを行っている。

【００２６】スケジューリング機能１０５におけるスケジューリング・アルゴリズムには、プライオリティ順やタイムシェアリング、またはそれらを組み合わせたものなどさまざまなタイプのものが考えられる。たとえば、図３は、プライオリティ順のスケジューリングアルゴリズムを実現した一例を説明する図である。この場合、スケジューリング機能１０５は、実行待キュー７７０を保持している。実行待キュー７７０は、優先順位毎に実行待状態のタスクをリスト上に保持し、その並び順は、待ち行列としてＦＩＦＯのキューとして管理している。最高優先順位実行待キュー７７１にリンクされたタスクの中で先頭のタスク７７４が次実行タスクに決定される。図３のスケジューリングアルゴリズムは、優先度＋待ち順となっている。実行可能なタスクは優先度毎に独立した待ち行列（ＦＩＦＯ）に並べられている。実行可能なタスク群の中で、もっとも優先度が高く、且つその待ち行列の先頭に位置するタスクが次実行タスクとなる。つまり、常に図３の左上のタスクが実行権を得ることになる（図３ではタスク１）。よって、図３のタスク１～タスク７における番号（数字）は実行される順番を表しているものでは無い。実行中のタスク（図のタスク１）が資源待ちなどの要因で実行可能タスク群から外されると、次にもっとも左上となるタスク（図ではタスク２）が実行権を得る。また、現在実行中のタスクよりも優先度の高いタスクが実行可能状態になった時は、そのタスクが実行権を得ることになる。

【００２７】また、別の例として、図４は、タイムシェアリングによるスケジューリングを実現した一例を示す図である。図４によれば、スケジューリング機能１０５は、実行待キュー７９０を保持している。実行待キュー７９０は、待ち行列としてＦＩＦＯのキューとして管理している。実行状態にある先頭のタスク７９１が一定時間経過した場合、そのタスクは待ち行列キューの最後尾に送られ、キューの２番目のタスク７９２が先頭になりそのタスクが次実行タスクに決定される。このように、実行待ちのタスクは時間によって順次切り替えられていく。スケジューリング・アルゴリズムとしては、これらを組み合わせたものや、さらに複雑な計算式に基づいて決定する方式などが考えられる。

【００２８】次に、以上のような基本構成を有するＲＴＯＳを（１）実機上と（２）ＵＮＩＸワークステーション上に適用するための機種依存部分１０６と、スケジューリング機能１０５に関して説明する。なお、共通部分１０３は（１）、（２）とも同一のプログラム・モジュールを使用している。

【００２９】（１）実機におけるＲＴＯＳ

ＣＰＵ２１として、モトローラ社製のＭＣ６８０００を搭載した実機２０に対して本実施形態のＲＴＯＳを動作させるものとする。図１における機種依存部分１０６（図２４におけるＲＴＯＳのハードウエア依存部分１０６ｂ）の構成要素について説明する。本実施形態では、メモリ（ＲＡＭ２３）上に、機種依存部分１０６がアクセスするタスク固有のコンテキスト情報（ハードウェア・レジスタ、プログラムカウンタ、タスクスタックなど）を格納するタスク・コントロール・ブロック（以下、ＴＣＢと記す）を作成する。アプリケーション・タスクはＲＴＯＳの初期化ファイルの中で定義され、コンパイル時に静的に配置されているものとする。

【００３０】❶タスク中断機能１０９：図５は、実機側におけるタスク中断機能の処理の流れを表すフローチャートである。タスクスイッチ制御機能１０８はタスク中断要求を受けるとタスク中断機能１０９を実行する。タスク中断機能１０９は、現在実行中のタスクのコンテキスト情報を、そのタスクのＴＣＢに保存する。具体的には、スタック上に保存されたプログラムカウンタの値を読み出し、読み出したプログラムカウンタの値、ハードウェア・レジスタの値およびスタックポインタ等をそのタスクのＴＣＢに書き込む（ステップＳ５０２、Ｓ５０３）。そして、当該ＴＣＢ内のタスクの状態を中断状態にセットして、中断処理を完了する（ステップＳ５０４）。

【００３１】❷タスク再開機能１１０：図６は、実機側におけるタスク再開機能１０９の処理の流れを表すフローチャートである。タスク再開要求を受けると、まずタスク再開機能１０９はタスク再開要求から再開すべきタスクのＩＤを獲得する（ステップＳ６０２）。そして、獲得したタスクＩＤに対応するタスクのＴＣＢのタスクの状態を実行中とする（ステップＳ６０３）。更に、中断中のタスクのコンテキスト情報を、そのタスクのＴＣＢからロードする（ステップＳ６０４）。具体的には、該当するＴＣＢに保存されているハードウェア・レジスタ、プログラムカウンタ、スタックポインタの値を読み出し、それぞれハードウエアレジスタにセットすることによって、以前に中断した状態からそのタスクを再開することができる。

【００３２】❸タスクスイッチ制御機能１０８：実行状態のタスクを切り替える。具体的には、タスク中断要求を受けて上述のタスク中断機能１０９を起動させ、実行状態だったタスクのコンテキスト情報を保存して中断状態にする。また、タスク再開要求を受けた場合は、上述のタスク再開機能１１０を起動させ、次の実行タスクのコンテキスト情報をロードして、中断状態のタスクを実

行状態へ切り替える。

【００３３】上記（１）～（３）は、機種依存部分１０６の構成要素であり、アセンブリ言語で書かれており、その中心となるのはタスクスイッチなどのコンテキスト操作を行う機能である。ＲＴＯＳのシステムコールの実体１０４およびスケジューリング機能１０５は共通部分１０３に含まれる。

【００３４】（２）シミュレーション環境におけるＲＴＯＳ

次に、シミュレーション環境における本ＲＴＯＳの実装例として、Sun Microsystems社の汎用ＵＮＩＸＯＳである、ＳｕｎＯＳ５．ｘ（ｘはマイナーバージョンをあらわす数字。以降、略してＳｕｎＯＳと呼ぶ）を搭載したワークステーション上で本ＲＴＯＳのシミュレータを動作させる場合を説明する。以下では、図１における機種依存部分１０６（シミュレーション環境側の機種依存部分であり、図２４の汎用ＯＳ依存部分１０６ａに対応する）の構成要素について説明する。なお、本実施形態では、ＳｕｎＯＳが提供するマルチスレッドライブラリを利用して、単一プロセス内でアプリケーション・タスクをスレッドとして動作させる。

【００３５】ＳｕｎＯＳでは、スレッドの中断、再開はライブラリが提供するシステムコールによって実現される。また、アプリケーション・タスクは、メイン・スレッドの初期化ルーチンの中でシステムコールにより生成されるものとする。

【００３６】❶タスク中断機能１０９：図７はシミュレーション機側におけるタスク中断機能の処理の流れを表すフローチャートである。タスク中断要求によりタスク中断機能が実行されると、スレッドライブラリが提供するthr_suspend()システムコールにより、スレッド化されたタスクの実行を中断状態にする（ステップＳ１２０２）。タスクのコンテキスト情報はＳｕｎＯＳによって適切に保存される。その後、ステップＳ１２０３において、当該タスクのＴＣＢにおける“タスクの状態”を“中断状態”にセットする。

【００３７】❷タスク再開機能１１０：図８は、シミュレーション機側のタスク再開機能の処理の流れを表すフローチャートである。タスク再開要求によりタスク再開機能が起動されると、実行を再開すべきタスクのタスクＩＤをタスク再開要求より獲得する（ステップＳ１３０２）。次に、実行再開すべきタスクのＴＣＢにおける“タスクの状態”を“実行中”とする（ステップＳ１３０２）。次に、スレッドライブラリが提供するthr_resume()システムコール（ステップＳ１３０２で得たタスクＩＤを引数とする）により、中断状態にあるスレッド化されたタスクを実行状態にすることができる（ステップＳ１３０４）。なお、タスクのコンテキスト情報はＳｕｎＯＳによって適切に復帰され、指定されたタスクを以前の中断位置から実行を再開することができる。

【００３８】❸タスクスイッチ制御機能１０８：実行状態のタスクを切り替える。具体的には、タスク中断要求を受けてタスク中断機能１０９を起動させ、実行状態だったタスク（スレッド）を中断状態とする。また、タスク再開要求を受けてタスク再開機能１１０を起動させ、次に実行するタスク（スレッド）を実行状態にする。この結果、実行状態となるアプリケーション・タスクは、この系の中で常に１つとなるため（あるタスクが１つ再開すると、実行中のタスクが１つ中断するので、常に１つのアプリケーションタスクしか動作していないことになる）、ＳｕｎＯＳによるスレッド・スケジューリングが起こっても、必ず目的とするタスク（スレッド）が実行される。このしくみにより、実機上で動く本ＲＴＯＳと同一の順番でタスクが実行されることが保証される。

【００３９】なお、上記❶～❸は、機種依存部分１０６の構成要素でＳｕｎＯＳのマルチスレッドライブラリを用いてＣ言語で書かれており、その中心となるのはタスクスイッチなどのコンテキスト操作を行う機能である。

【００４０】以上説明したように、第１の実施形態のＲＴＯＳによれば、ハードウェアに依存する部分を必要最小限となるように切りだし、それ以外の部分は様々なプラットフォームで共通に利用できるようにＣ言語で書かれており、ハードウェアの違いによる影響が極力少なくなるような構造を有する。

【００４１】すなわち、ハードウェアに依存する部分とは、❶タスクの実行を中断する機能と、❷中断されたタスクの実行を再開する機能と、これらの機能を用いて❸タスクスイッチの制御をする機能であり、これらの機能を実現するプログラムモジュールは対象とするＭＰＵや汎用ＯＳによって書き換える必要がある。

【００４２】一方、タスク制御やタスク間の同期・通信機能やタスクスイッチのスケジューリング機能等、その他すべてのＲＴＯＳの機能部分、つまり、ＲＴＯＳのシステムコールを提供する階層は、すべてＣ言語により記述されているハードウェアに依存しない階層である。このように、タスクのコンテキスト制御等ごく限られた機能で実現する機種依存部分と、システムコール本体を実現する共通部分が完全に切り分けられた構造を持つことにより、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことが可能となる。

【００４３】従って第１の実施形態によれば、ＲＴＯＳの振舞に関わるシステムコールとスケジューリング機能などは、機種に依存しない共通部分として流用し、最小限に抽出されたプラットフォームに依存するプリミティブな機能のみを実装するだけで、実機とシミュレーション環境などプラットフォームが異なっても、アプリケーションに同一の振舞をさせることができる。また、機種依存部分が少なく限られた部分に集中しているので、移植作業を容易に行える。

【００４４】［第２の実施形態］以下、第２の実施形態を詳細に説明する。第２の実施形態では、第１の実施形態で説明したＲＴＯＳにおいて、機種依存部分１０６内に動的にタスクを生成／削除する機能を追加したものである。すなわち、第１の実施形態では、タスクの動的な生成／削除を行わない（或いは、行う必要がない）システムを示したが、第２の実施形態では動的なタスクの生成／削除を行えるように拡張したシステムを示す。

【００４５】（１）実機側のＲＴＯＳ

第１の実施形態と同様に、モトローラ社製のＭＣ６８００００を搭載した実機に対して本ＲＴＯＳを動作させるものとする。図９は、第２の実施形態による実機側のＲＴＯＳの構成を示すブロック図である。図９のＲＴＯＳは、図１で示すＲＴＯＳの基本的な構成に、タスク生成機能１１２ｂと、タスク削除機能１１３ｂを加えた構成となっている。また、１１４ｂはタスク固有のコンテキスト情報（ハードウェア・レジスタ、プログラムカウンタ、タスクスタックなど）を格納するメモリ（ＲＡＭ２３）上の領域（ＴＣＢ）である。

【００４６】❶タスク生成機能１１２ｂ：タスク生成機能１１２はタスクのコンテキスト情報を生成する。ここで、タスクのコンテキスト情報とは、タスクのある瞬間のハードウェア・レジスタ（ＣＰＵ内部のレジスタ）、プログラムカウンタ、及びタスク・スタックの内容であり、タスク毎に固有の情報である。スタックポインタや、プログラムカウンタなどのコンテキストの内容は適切な値に初期化される。コンテキスト情報はあらかじめ用意されたメモリ上の空き領域に生成される（タクス情報１１４ｂ）。

【００４７】図１０は実機側のタスク生成機能の処理を説明するフローチャートである。タスク生成要求によってタスク生成機能１１２が起動すると、まず、ステップＳ３０２で、新たなＴＣＢを生成するための空きメモリ領域がＲＡＭ２３に存在するかを確認する。ここで空きメモリ領域が存在しなければステップＳ３０４でエラー処理を行う。すなわち、システムコールの戻り値は、そのシステムコールの実行後の状態（成功・失敗）を示しているため、図１０のステップＳ３０４の処理では「メモリ不足」を示すエラー番号を戻り値としてセットする処理を行う。一方、空きメモリ領域が存在するならば、ステップＳ３０３へ進み、空きメモリ領域から１つ分のＴＣＢを確保する。そしてステップＳ３０４において、確保したＴＣＢのスタックポインタや、プログラムカウンタ等を適切な値に初期化する。

【００４８】❷タスク削除機能１１３ｂ：図１１は実機側のタスク削除機能の処理を表すフローチャートである。タスク削除機能１１３は、タスク削除要求に応じて該当するタスクのコンテキスト情報を削除する。具体的には、そのコンテキスト情報を格納しているメモリ上の領域１１４を開放し、未使用状態とする（ステップＳ４０２）。

【００４９】（２）シミュレーション環境におけるＲＴＯＳ

第１の実施形態と同様、シミュレーション環境における実装例として、ＳｕｎＯＳを搭載したワークステーション上でＲＴＯＳのシミュレータを動作させるものとする。

【００５０】図１２は、第２の実施形態によるシミュレーション側のＲＴＯＳの構成を示すブロック図である。図１２のＲＴＯＳでは、図１で示すＲＴＯＳの基本的な構成に、タスク生成機能１１２ａと、タスク削除機能１１３ａが加わっている。１１４ａはタスク固有のコンテキスト情報で、ＳｕｎＯＳのシステム内部で管理されているメモリ上の領域である。

【００５１】❶タスク生成機能１１２ａ：タスク生成機能１１２ａは、タスク生成要求に応じて、スレッドライブラリが提供するthr_create()システムコールを用いてタスクをスレッドとして生成する。スレッドのコンテキスト情報はＳｕｎＯＳが管理する。つまり、図１２におけるタスク情報１１４ａはＳｕｎＯＳ内部で管理される領域である。なお、このシステムコールはスレッドを実行状態で生成するか、中断状態で生成するかをフラグにより指定できるが、本実施形態では、中断状態で生成し、タスク再開機能１１０によって実行を開始させるものとする。

【００５２】図１３はシミュレーション側のタスク生成機能の処理手順を表すフローチャートである。まず、ステップＳ１００２において、空きメモリがあるかを判定し、空きメモリが無ければステップＳ１００４で、図１０のステップＳ３０４と同様のエラー処理を行う。空きメモリがあればステップＳ１００３へ進み、空きメモリ領域からＴＣＢの１つ分のメモリ領域を確保する。そして、ステップＳ１００５にてthr_create()システムコールを呼び出して、中断状態のスレッドを生成する。その後、ステップＳ１００６にて、当該スレッドのＩＤをＴＣＢに登録する。なお、本ＲＴＯＳでは、初期化時にワークスペースと呼ばれるメモリ領域を一括して確保する。そして、ＴＣＢなどのシステム管理情報は、すべてこのワークスペース内に取られる。従って実機とシミュレーション機で同じサイズのメモリ領域を（初期化時に）確保しておけば、実機／シミュレーション機間における所有メモリ領域（サイズ）は一致するので、実機と同様の空きメモリチェックを行なえる。また、ＲＴＯＳの共有部分では、タスクＩＤでタスクの実行、中断を指示するが、タスクＩＤ、スレッドＩＤ共にＴＣＢの中に保存されている。ここで、ＴＣＢは機種依存部分に存在するため、タスクＩＤとスレッドＩＤを１対１に対応づけることが出来る。また、図１３のステップＳ１００５ではthr_Create()システムコールを用いてタスク生成を行なうが、このシステムコールによって生成されたスレ

ッドのＩＤが当該関数から返される。従って、このスレッドＩＤをタスクＩＤに対応付けることになる。

【００５３】❷タスク削除機能１１３ａ：スレッドライブラリが提供するthr_delete()システムコールにより、スレッド化されたタスクを削除する。図１４はシミュレーション側のタスク削除機能の処理を示すフローチャートである。タスク削除要求を受け付けると、当該タスク削除要求に含まれるスレッドＩＤを獲得し、これを引数としてthr_delete()システムコールを行う（ステップＳ１１０２）。その後、当該スレッドに対応するＴＣＢが使用していたメモリ領域を開放し、未使用状態とする。

【００５４】［第３の実施形態］第３の実施形態では、第１，２の実施形態におけるＲＴＯＳにおいて、更に割込みを管理する機能を追加した場合を説明する。

【００５５】（１）実機におけるＲＴＯＳ

第１および第２の実施形態と同様に、モトローラ社製のＭＣ６８０００を搭載した実機に対して本ＲＴＯＳを動作させるものとし、以下では第３の実施形態で新たに追加された割込みを管理する機能について説明する。

【００５６】図１５は、第３の実施形態による実機側のＲＴＯＳの構成を示すブロック図である。図９で示した構成に、割込み管理機能１２０、割込みコントローラ１２１、割込みハンドラ１２２が加えられている。なお、割り込みコントローラとは、ハードウェア割り込みの制御を行うハードウェアである。ＡＴ互換機などでは８２５９ＡというＬＳＩとして知られている。ハードウェア割り込みはこの割り込みコントローラを経由してＣＰＵに伝えられ、発生した割り込みに対応した割り込みハンドラが起動される。図１５における割り込みコントローラおよび割り込みハンドラの役割は、ハードウェア割り込みが生じたことを、当該ＲＴＯＳの割り込み管理機能に伝えることである。

【００５７】❶割込み管理機能１２０：割込み開始要求を受けて、タスク中断機能１０９ｂを呼び出し、実行中のタスクのＴＣＢ１１４にコンテキストの保存を行う。割込みが多重に起こった場合は、システムスタック１２３上にレジスタを積み上げていくことにより、多重割込みに対応する。割込み終了要求を受けて、スケジューリング機能１０６を呼び出した後、タスク再開機能１１５を呼び出して次タスクのＴＣＢ１１４からコンテキストの復帰を行う。多重割込みからの復帰の場合は、システムスタック１２３上からコンテキストの復帰を行う。

【００５８】図１６は、実機側の割込み処理を説明するフローチャートである。ハードウェア割込みが発生すると、割込みハンドラ１２２の先頭で割込み管理機能１２０に対して割込み開始要求を出す（ステップＳ８０２）。なお、割り込みハンドラの先頭とは、ハンドラルーチンが最初に処理するステップのことである。割込み開始要求を受けた割込み管理機能１２０では、まず、当該割込み要求が多重割込みか否かを判定する。多重割込みであった場合は、ステップＳ８０５へ進み、システムスタック１２３にレジスタを待避させる。一方、多重割込みでなければ、ステップＳ８０３からステップＳ８０４へ進み、タスク中断機能１０９を起動させる。なお、システムの属性値として、（多重）割り込み回数をカウントするカウンタが存在する。割り込み管理機能にて割り込み処理を開始する時にカウンタをインクリメントし、割り込み処理を終了する直前にカウンタをデクリメントする。多重割り込みか否かの判断はこのカウンタの値を元に行われる。また、ステップＳ８０５においてシステムスタック１２３に待避されるのは、処理中の割り込みに関わるレジスタ値である。例えば、現在処理中の割り込みＡが、それより優先度の高い割り込みＢにより中断された瞬間のレジスタ値をシステムスタックに保存し、高優先度の割り込みＢの処理が終わった後に、割り込みＡの処理を再開できるようにする。

【００５９】次にステップＳ８０６において、割込みハンドラ１２２の本体を実行させる。割込み処理本体を実行した後、割込みハンドラを終了する直前に、割込み管理機能１２０に対して割込み終了要求を出す（ステップＳ８０７）。なお、ステップＳ８０６では、割り込みが発生した要因に基づくＩ／Ｏ処理など、ハンドラの本来の処理が行われる。例えば、ＲＳ－２３２Ｃなどのシリアル通信では、シリアルポートへデータがセットされた時に受信割り込みが発生し、対応する割り込みハンドラが起動されるが、割り込みハンドラでは受信したデータを読み込むためのＩ／Ｏ処理と読み取ったデータをバッファなどにつめる等の処理とが行なわれる。これが、ここで言うハンドラの本体の処理になります。Ｓ８０７はステップＳ８０２と同様に、割り込み管理機能を呼び出すという動作そのものであり、具体的な処理としては、図１６においてその下に描かれる破線の中の処理が行われることになる。

【００６０】割込み終了要求を受けた割込み管理異能１２０は、ステップＳ８０８において、当該割込みが多重割込みか否かを判定する。多重割込みであった場合は、ステップＳ８１１へ進み、システムスタック１２３からレジスタを復帰させて、ステップＳ８１２へ進む。一方、多重割込みでない場合は、ステップＳ８０９へ進み、スケジューリング機能１０５を呼び出し、次に再開すべきタスクのタスクＩＤを獲得する。そして、ステップＳ８１０において、タスク再開機能１１０を起動する。そしてステップＳ８１２において、ハードウエア割り込み終了、すなわち割り込み復帰命令を実行して本処理を終える。割り込み復帰命令では、その割り込みレベルを解除し（一つ前のレベルに戻す）、システムスタック上に保存されているＰＣ（プログラムカウンタ）や、ＳＲ（ステータスレジスタ）などを読み出して、割り込まれた個所へ復帰を行なう。なお、その他のレジスタの復帰は事前に行なう必要がある。なお、ステップＳ８０

６の割り込みハンドラ本体の処理の中では、本ＲＴＯＳのシステムコールを実行する可能性がある。そのような場合、システムコールの実行によってシステムの状態が変化し、実行中だったタスクが資源待ちになったり、優先度の高いタスクが実行権を獲得するような状況が生じる。通常はシステムコールの終了の直前にスケジューリングされるが、割り込み処理中はタスクスイッチすることができないので、割り込み処理の終了の直前にスケジューリングを行う必要がある。従って、ステップＳ８１０で再開されるタスクは、割り込みの直前に実行されていたタスクとは限らない。

【００６１】（２）シミュレーション環境におけるＲＴＯＳ

第１の実施形態と同様、シミュレーション環境における実装例として、ＳｕｎＯＳを搭載したワークステーション上で動作する本ＲＴＯＳのシミュレータを動作させるもので、さらに追加した割込み管理機能について説明する。

【００６２】図１７は、第３の実施形態によるシミュレーション装置側のＲＴＯＳの構成を示す図である。図１７では、シミュレーション側のＲＴＯＳの基本的な構成に割込み管理機能１３２を加えた構成が示されている。

【００６３】①割込み管理機能（スーパーシグナルスレッド）１３２：図１７に示される割込み管理機能１３２は、特定のシグナルを用いて、複数のハードウェア割込みをシミュレートするしくみである。これは、アプリケーション・タスクとバインドするスレッド群とは独立して動く、割込み管理専用のスーパーシグナルスレッドとして実現する。１３１は汎用ＯＳからシグナルを受けた時に呼ばれるシグナルハンドラであり、シグナルハンドラ１３１はスーパーシグナルスレッド１３２に事象の発生を通知する。スーパーシグナルスレッド１３２は、多重割込みをシミュレートするために独自のスタック１３３で割込みレベルの管理をする。１つの割込みハンドラは１つのスレッド（ハンドラスレッド）１３４として生成され起動される。つまり、多重割込みが発生した場合は、多重のネスト回数と同数のハンドラスレッド１３４が生成されることになる。また、割込みレベル制御およびハンドラスレッド１３４の管理はスーパーシグナルスレッド１３２が行い、多重割込みのシミュレーションを行う。

【００６４】図１８は割込みシグナル等の事象が発生した場合の割込み管理機能１３２の処理の流れを示したフローチャートである。ステップＳ１６０１において、シグナルハンドラ１３１が割込みの発生を検出すると、シグナルハンドラ１３１はスーパーシグナルスレッド１３２へ割込みイベントの発生を通知する。

【００６５】割込みイベントの通知を受けたスーパーシグナルスレッド１３２は、当該割込みが多重割込みか否かを判定する。多重割込みであった場合は、ステップＳ１６０２からステップＳ１６０４へ進み、実行中のハンドラスレッドを中断状態にしてスタック１３３にスタックする。一方、多重割込みでなければ、ステップＳ１６０２からステップＳ１６０３へ進み、タスク中断機能１０９を起動させて、実行中のタスクを中断する。次に、ステップＳ１６０５において、対応するハンドラスレッドを生成する。

【００６６】ステップＳ１６０６において、生成されたハンドラスレッド１３４が起動され、割込みハンドラ本体が実行される。次にステップＳ１６０８において、多重割込みか否かを判定し、多重割込みであったならばステップＳ１６１０へ進む。ステップＳ１６１０においては、中断中のハンドラスレッドをスタックから取り出し、実行状態にする。なお、中断中のハンドラ用スレッドを再開（実行状態にする）するのは、スーパーシグナルスレッド内のステップＳ１６１０で行われるが、実際にそのスレッドが再開するタイミングは汎用ＯＳのスケジューリングに任される。通常、スーパーシグナルスレッドは最も優先度の高いスレッドとして生成するので、一連の割り込み処理はＳ１６１０、Ｓ１６１２まで流れ、終了する。その後、Ｓ１６１０で実行状態にされたスレッドが（汎用ＯＳによって）再開される。

【００６７】シミュレーション環境においては、割り込み管理機能もシミュレーションする必要があるため、多重割り込みのシミュレーションを行うには、割り込みハンドラをも中断／再開してコントロールしなければならない。そのため、汎用ＯＳが提供するスレッドを生成して、そのスレッドでハンドラを実行し、中断／再開などの操作はスーパーシグナルスレッドが管理する。図中ステップＳ１６０６はハンドラ用のスレッドを生成するステップで、ステップＳ１６０７の割り込みハンドラ本体は、割り込みの要因に対する具体的な処理を記述した関数（もしくはブロック）である。

【００６８】ステップＳ１６０８において多重割込みでなかった場合（多重割込みで最後の割込み処理を得た場合も含む）、ステップＳ１６０９へ進み、スケジューリング機能１０５を読み出し、ステップＳ１６１１でタスク再開機能１１０を呼び出す。ここでシミュレーション側の割り込みはたとえば次のようにして発行させる。シミュレーション環境での割り込みをハンドリングするためには、独立して動くスレッドもしくはプロセスから、スーパーシグナルスレッドへ通知しなくてはならない。タイマ割り込みなどは、完全に独立して動くタイマ・スレッドを生成して、定期的にスーパーシグナルスレッドへ通知する。プロセス外からの割り込み要因の通知は、汎用ＯＳが提供するシグナルハンドラを利用する。ＵＮＩＸを例に説明すると、あらかじめＲＴＯＳのプロセスでシグナルハンドラを登録しておき、ｋｉｌｌコマンドなどで別のプロセスから当該プロセスにシグナルを送る。そして起動されたシグナルハンドラの中からスーパ

ーシグナルスレッドへ割り込みを通知する。

【００６９】以上のように第３の実施形態によれば、ＲＴＯＳを用いたアプリケーションのシミュレーションにおいて割込み処理も扱えるようになる。

【００７０】なお、上記各実施形態では、シミュレーション環境としてＳｕｎＯＳを用いた場合を説明したが、他のＯＳ、例えば、Microsoft社のＷｉｎｄｏｗｓ９５、ＷｉｎｄｏｗＮＴなどのＷｉｎ３２－ＡＰＩで提供されるマルチスレッド機能を用いて、同様なＲＴＯＳシミュレータを構築可能である。すなわち、上記実施形態で説明したＲＴＯＳは、第１乃至第３の実施形態で紹介したＳｕｎＯＳのシステムコール（thr_create(), thr_delete(), thr_suspend(), thr_resume()など）と同等なシステムコールを提供している汎用ＯＳ上で動作するシミュレータとして簡単に移植が可能である。

【００７１】［第４の実施形態］図１９は、第４の実施形態によるＲＴＯＳの構造を示す図である。図１９では、図１に示したＲＴＯＳ（シミュレーション機側）にデバッグ情報を取得する機能が付加されている。

【００７２】ＲＴＯＳのシステムコールの実体は共通部分１０３にあり、各システムコールはすべて共通のシステムコール・エントリポイント１８０１を経由する構造になっている。そして、このシステムコール・エントリポイント１８０１は、トレース情報記録機能１８０２を有する。トレース情報記録機能１８０２は、システムコールが発行される毎に図２０に示すような実行トレース情報９０１を生成する。すなわち、トレース情報記録機能１８０２は、発行されたシステムコールを取り出し、呼び出しタスクＩＤ９０２、システムコール実行情報９０３、タイムスタンプ９０４などの情報を取り出し、実行トレース情報９０１として一時的に保存する。ここで、呼び出しタスクＩＤ９０２は、タスクＩＤ又は、ＯＳ、割込みの識別子を特定する情報である。システムコール実行情報９０３は、システムコール名（ＩＤ）およびパラメータ群を含む。更にタイムスタンプ９０４は、当該システムコール発行時のタイマカウンタの値を示す。

【００７３】また、デバッグ情報転送機能１８０３は、保存した実行トレース情報９０１をグラフィカルに表示する可視化ツール（タスクスイッチ・ビューア）１８０６などの外部モジュールに転送する。

【００７４】トレース情報記録機能１８０２は１回のシステムコールで図２０に示す構造を持つ実行トレース情報を１つ記録するので、アプリケーションの一連の動作の履歴は大きな記憶容量を必要とする場合がある。加えて、一般的に実機環境ではメモリ領域などの資源が限られているため、データの記録方法になんらかの工夫が必要となる。このような問題に対処する方法として、例えば、リングバッファを用いた記録方法などが考えられる。リングバッファは、連続して生成される膨大なデータの中で最近のデータだけを効率良く記録する仕組みで、記録するデータ数がバッファサイズを超える場合に、最も古いデータから上書きしていく手法である。図１９では、トレース情報記録機能１８０２で生成された実行トレース情報９０１をリングバッファ１８０７に保存していく例が示されている。リングバッファ１８０７はメモリ上に配置され、記録開始位置を指し示すＳｔａｒｔポインタ１８０８と、記録終了位置を指し示すＥｎｄポインタ１８０９を用いてデータの記録、取り出し作業を行う。

【００７５】図２１は、トレース情報記録機能１８０２の処理の流れを示したフローチャートである。まず、システムコールの発行等により、ステップＳ１９０１において実行トレース情報が生成されると、ステップＳ１９０２において、リングバッファ１８０７のＥｎｄポインタ１８０９が示す位置に当該実行トレース情報を書き込む。次にステップＳ１９０３において、有効データ数がバッファサイズ以上になっているかを判定する。有効データ数がバッファサイズより小さければステップＳ１９０５へ進み、有効データ数を１つインクリメントする。一方、有効データ数がバッファサイズ以上となっている場合は、Ｓｔａｒｔポインタ１８０８をインクリメントする。このときＳｔａｒｔポインタ１８０８の位置がオーバーランした場合は先頭に戻す。そして、ステップＳ１９０６において、Ｅｎｄポインタ１８０９の位置をインクリメントし、オーバーランした場合は先頭に戻す。なお、リングバッファは通常の物理的に線形なバッファを用いて、論理的にリング状に見えるような仕組みを持つものである。Start位置やEnd位置を指し示すポインタは、操作の度にインクリメントされるが、いづれ物理バッファの最後尾に到達する。さらにインクリメントするとバッファをオーバーランしてしまうので、その時点でポインタをバッファの先頭にリセットする。この仕組みにより、論理的にリング状に見せかけることができる。

【００７６】また、デバッグ情報転送機能１８０３は、図１における共通部分１０３の内部に実装されるが、物理的な通信経路はターゲットシステムに依存する。そのため、ターゲットシステムが備えている通信手段を利用するために、ハードウエア依存部分１０６にデバッグ通信モジュール１８０５を具備する。デバッグ通信モジュール１８０５は、バッファのｒｅａｄ／ｗｒｉｔｅを行うＩ／Ｆを提供する。最下層のレイヤとしては具体的には、シリアル、ソケット、共有メモリ、ＩＣＥ、ＲＯＭエミュレータなどが考えられる。

【００７７】図２２はデバッグ情報転送機能１８０３において、先の例で示したリングバッファを用いた場合の処理の手順を示したフローチャートである。まず、デバッグ情報転送機能１８０３は、ステップＳ２００２において有効データ数が１以上であるか否かを判断する。有効データ数が１以上であれば、転送すべき実行トレース

情報が存在すると判断し、ステップＳ２００３へ進む。ステップＳ２００３では、リングバッファからデータを１つ取り出す。そして、ステップＳ２００４において、取り出した実行トレース情報をデバッグ通信モジュール１８０５を介して外部モジュール（例えば可視化ツール１８０６）へ転送する。そして、ステップＳ２００５において有効データ数を１つ減少させて、ステップＳ２００２へ戻る。

【００７８】この様にして転送したトレース情報をグラフィカルに表示する表示装置として、図２３にタスクスイッチ・ビューア１８０６の概観の一例を示す。

【００７９】この様に、ＲＴＯＳのデバッグ情報として有効なシステムコールやタスクスイッチの履歴、システム管理情報などを取得する手段を共通部分で実装することにより、異なるプラットフォーム間でも統一された思想でデバッグを行うことができる。

【００８０】以上のように、第４の実施形態のＲＴＯＳは、ハードウェアに依存する部分を必要最小限となるように切り出し、それ以外の部分はさまざまなプラットフォームで共通に利用できるようにＣ言語で書かれており、ハードウェアの違いによる影響が少なくなるような構造を持つものであり、システムコールはすべて共通のエントリポイントを経由してコールされ、システムコール実行後にスケジューリング機能が呼び出される構造を持つ。そして、この共通エントリポイントにおいて（１）呼びだしタスクＩＤと（２）システムコール実行情報と（３）タイムスタンプなどの情報を保存するトレース情報記録機能と、保存したトレース情報をタスクスイッチ・ビューアなどの外部モジュールに転送する手段とを設けたことにより、プラットフォームに依存しないデバッグ情報を提供することが可能である。

【００８１】以上説明したように、上記各実施形態によれば，次のような効果が期待できる。すなわち、

（１）プラットフォーム非依存性

アプリケーション・タスクをまったく修正せずに、実機と汎用開発マシンなど異なるプラットフォーム上で同一の振る舞いをさせることができる。

（２）移植の容易性

機種依存部分が最小限に切り出されているので、他のプラットフォームへの移植が容易に実現できる。

（３）プラットフォーム非依存なデバッガ

ハードウェアに依存する物理的通信を行うモジュールだけ実装すれば、様々なプラットフォームで、同一のデバッガが利用できる。

（４）デバッグ情報の統一化

取得するデバッグ情報はプラットフォームに依存しない情報なので、異なるプラットフォーム間でも統一思想でデバッグを行うことができる。

【００８２】なお、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）が記憶媒体に格納されたプログラムコードを読出し実行することによっても、達成されることは言うまでもない。

【００８３】この場合、記憶媒体から読出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。

【００８４】プログラムコードを供給するための記憶媒体としては、例えば、フロッピディスク，ハードディスク，光ディスク，光磁気ディスク，ＣＤ－ＲＯＭ，ＣＤ－Ｒ，ＤＶＤ，磁気テープ，不揮発性のメモリカード，ＲＯＭなどを用いることができる。

【００８５】また、コンピュータが読出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているＯＳ（オペレーティングシステム）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【００８６】さらに、記憶媒体から読出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【００８７】

【発明の効果】以上説明したように、本発明のマルチタスク制御方法によれば、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことが可能となる。

【００８８】また、本発明によれば、あらゆるプラットフォーム上で同一のインターフェース仕様を持ち、アプリケーションが同一の動作を行うことを可能とするＲＴＯＳ上のデバッグ環境（タスクスイッチ・ビューア等）において、あらゆるプラットフォーム上で統一的なデバッグ情報を収集することが可能となる。

【００８９】

【図面の簡単な説明】

【図１】第１の実施形態におけるＲＴＯＳの基本的な構成図である。

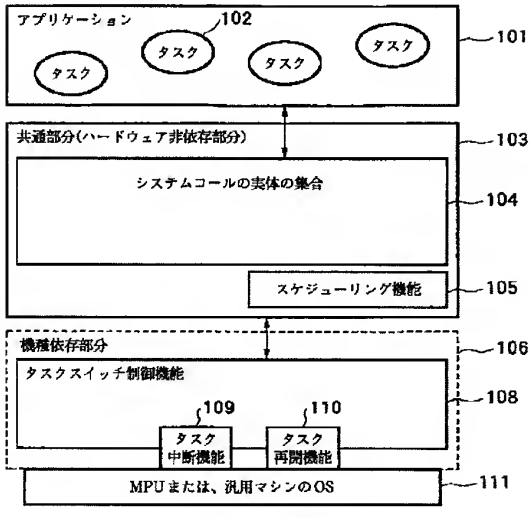
【図２】システムコール発行からタスクスイッチまでの、共通部分による処理の手順を表したフローチャートである。

【図３】プライオリティ順のスケジューリングアルゴリズムを実現した一例を説明する図である。

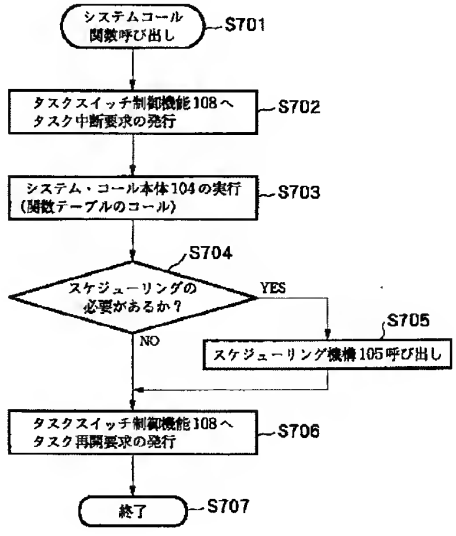
【図４】タイムシェアリングによるスケジューリングを

実現した一例を示す図である。
【図５】実機側におけるタスク中断機能の処理の流れを表すフローチャートである。
【図６】実機側におけるタスク再開機能１０９の処理の流れを表すフローチャートである。
【図７】シミュレーション機側におけるタスク中断機能の処理の流れを表すフローチャートである。
【図８】シミュレーション機側のタスク再開機能の処理の流れを表すフローチャートである。
【図９】第２の実施形態による実機側のＲＴＯＳの構成を示すブロック図である。
【図１０】実機側のタスク生成機能の処理を説明するフローチャートである。
【図１１】実機側のタスク削除機能の処理を表すフローチャートである。
【図１２】第２の実施形態によるシミュレーション側のＲＴＯＳの構成を示すブロック図である。
【図１３】シミュレーション側のタスク生成機能の処理手順を表すフローチャートである。
【図１４】シミュレーション側のタスク削除機能の処理を示すフローチャートである。
【図１５】第３の実施形態による実機側のＲＴＯＳの構成を示すブロック図である。
【図１６】実機側の割込み処理を説明するフローチャートである。
【図１７】第３の実施形態によるシミュレーション装置側のＲＴＯＳの構成を示す図である。
＊【図１８】シグナル等の事象が発生した場合の割込み管理機能１３２の処理の流れを示したフローチャートである。
【図１９】第４の実施形態によるＲＴＯＳの構造を示す図である。
【図２０】実行トレース情報のデータ構成例を示す図である。
【図２１】トレース情報記録機能１８０２の処理の流れを示したフローチャートである。
【図２２】デバッグ情報転送機能１８０３において、先の例で示したリングバッファを用いた場合の処理の手順を示したフローチャートである。
【図２３】タスクスイッチ・ビューア１８０６の概観の一例を示す図である。
【図２４】第１の実施形態における実機とシミュレーション環境を提供するコンピュータ装置を示す図である。
【符号の説明】
１０１　アプリケーション
１０２　アプリケーション・タスク
１０３　ＲＴＯＳの共通部分
１０４　ＲＴＯＳシステムコールの実体の集合
１０５　スケジューリング機能
１０６　ＲＴＯＳの機種依存部分
１０８　タスクスイッチ制御機能
１０９　タスク中断機能
１１０　タスク再開機能

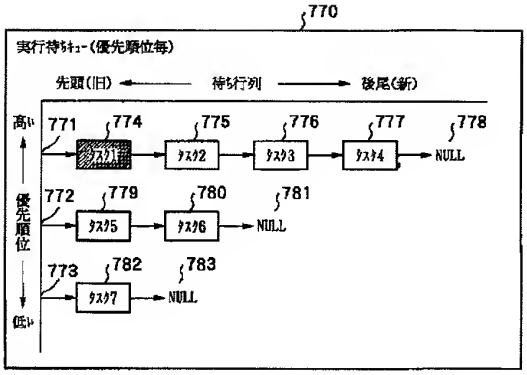
【図１】



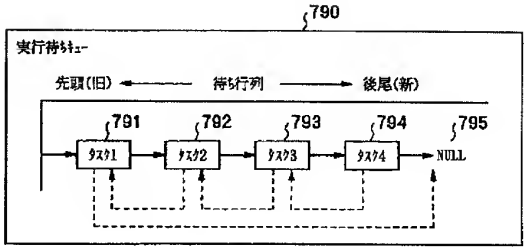
【図２】



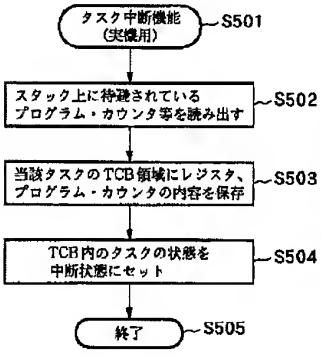
【図3】



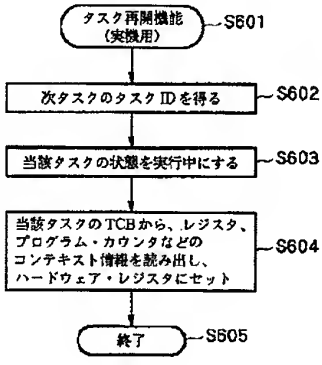
【図4】



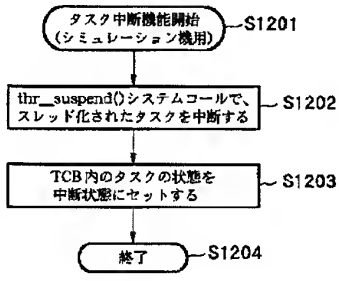
【図5】



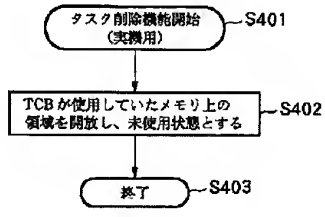
【図6】



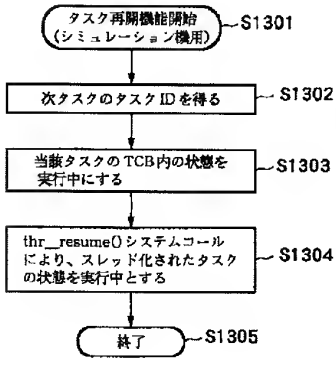
【図7】



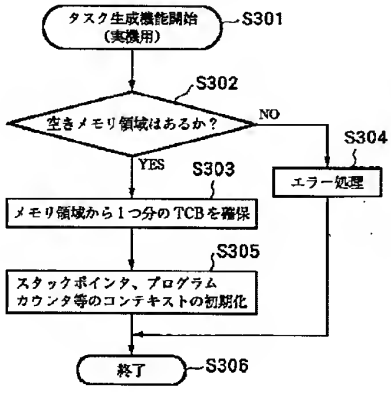
【図11】



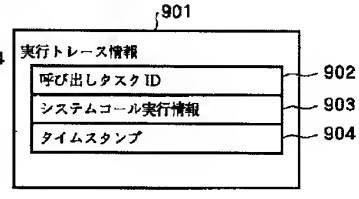
【図8】



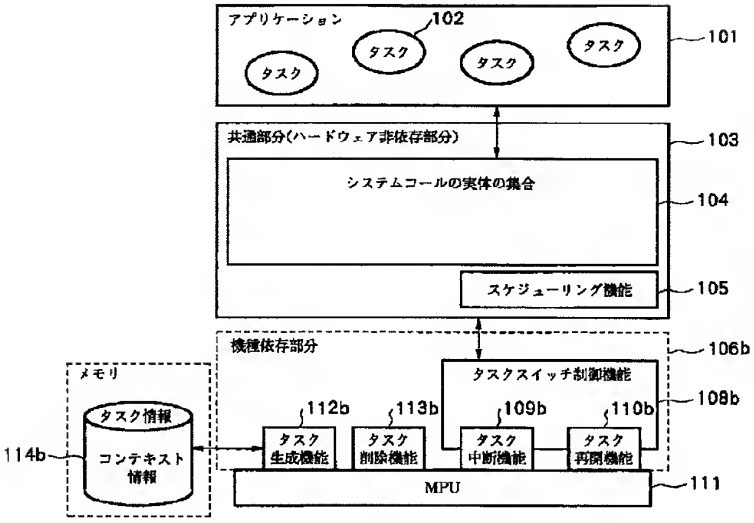
【図10】



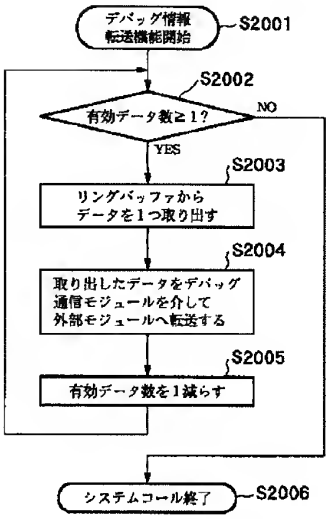
【図20】



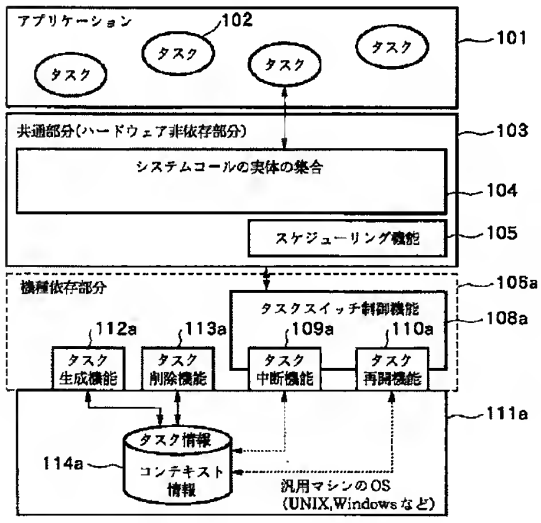
【図9】



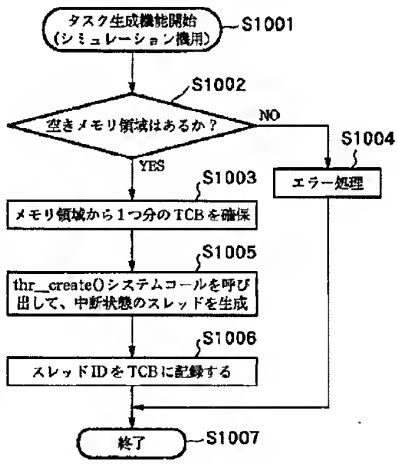
【図22】



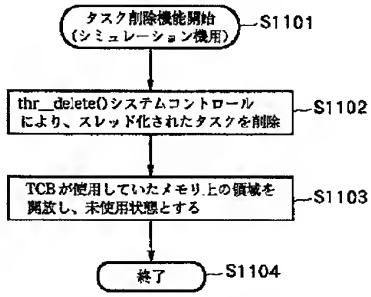
【図12】



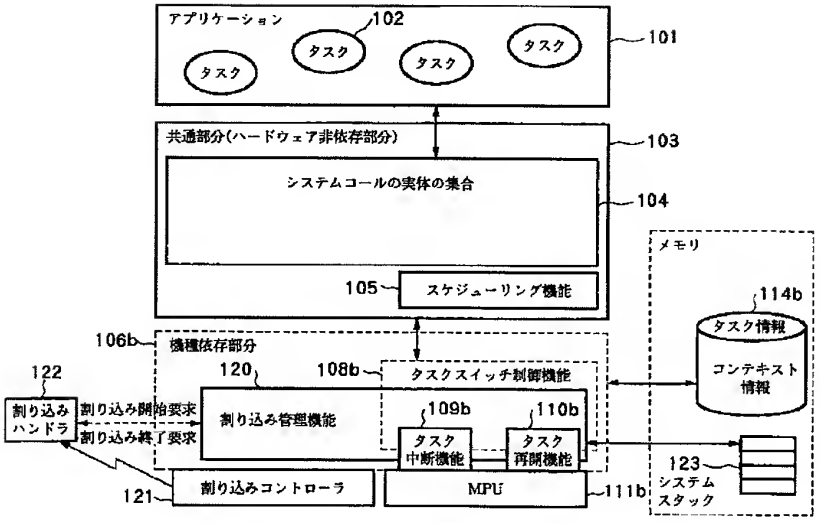
【図13】



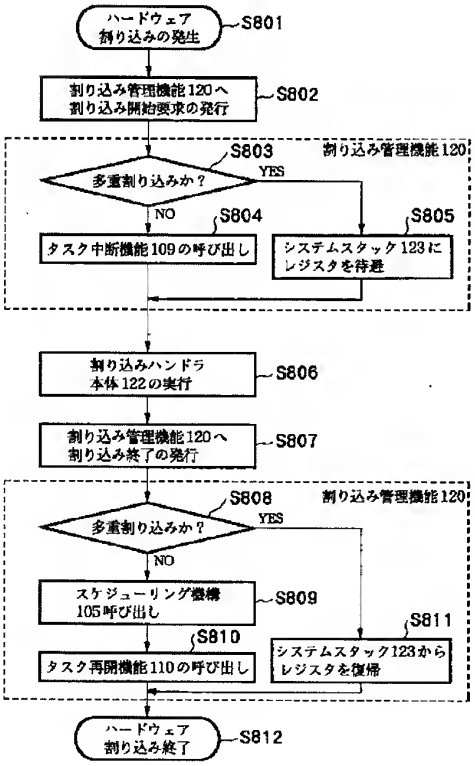
【図14】



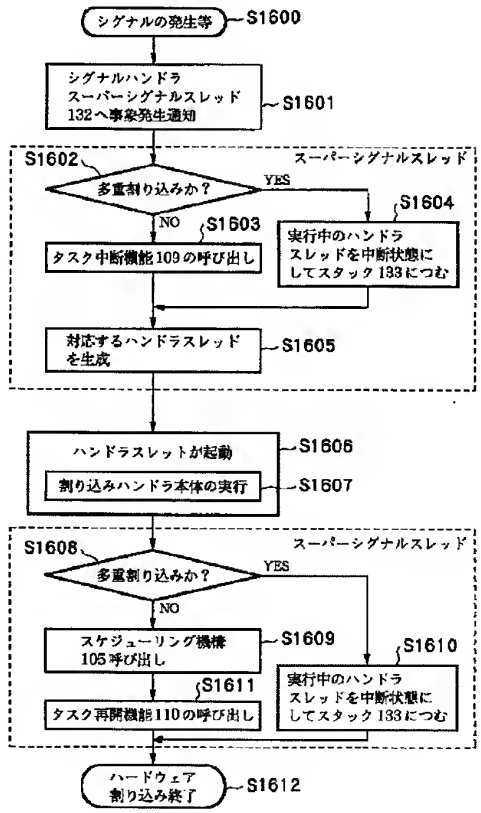
【図１５】


アプリケーション
102
タスク
タスク
タスク
タスク
101
共通部分(ハードウェア非依存部分)
103
システムコールの実体の集合
104
105
スケジューリング機能
メモリ
114b
タスク情報
コンテキスト情報
106b
機種依存部分
122
120
108b
タスクスイッチ制御機能
割り込みハンドラ
割り込み開始要求
割り込み終了要求
割り込み管理機能
109b
110b
タスク中断機能
タスク再開機能
123
システムスタック
割り込みコントローラ
MPU
111b
121

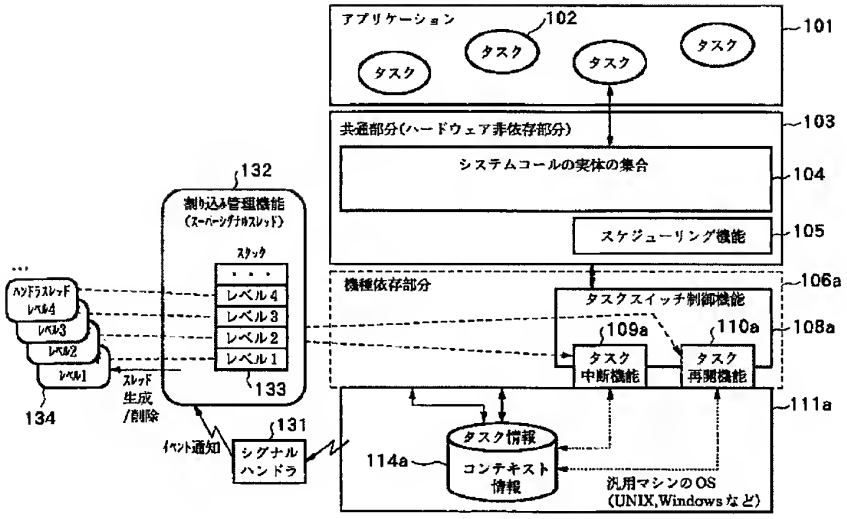
【図１６】


ハードウェア割り込みの発生
S801
割り込み管理機能120へ割り込み開始要求の発行
S802
割り込み管理機能120
S803
多重割り込みか？
YES
NO
S804
タスク中断機能109の呼び出し
S805
システムスタック123にレジスタを待避
割り込みハンドラ本体122の実行
S806
割り込み管理機能120へ割り込み終了の発行
S807
割り込み管理機能120
S808
多重割り込みか？
YES
NO
スケジューリング機構105呼び出し
S809
S810
タスク再開機能110の呼び出し
S811
システムスタック123からレジスタを復帰
ハードウェア割り込み終了
S812

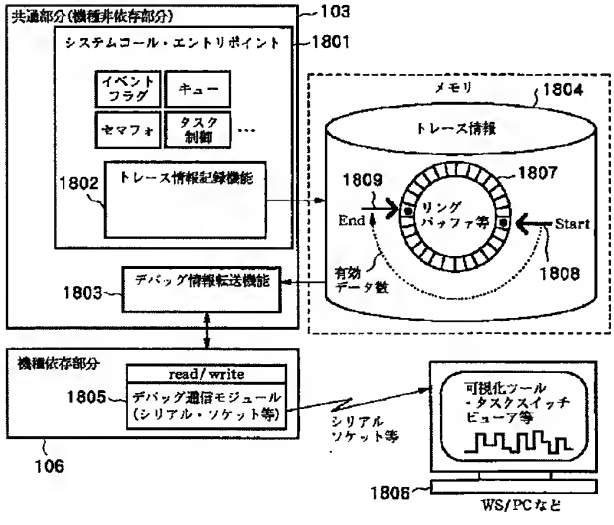
【図１８】


シグナルの発生等
S1600
シグナルハンドラ スーパーシグナルスレッド132へ事象発生通知
S1601
スーパーシグナルスレッド
S1602
多重割り込みか？
YES
NO
S1603
タスク中断機能109の呼び出し
S1604
実行中のハンドラスレッドを中断状態にしてスタック133につむ
対応するハンドラスレッドを生成
S1605
ハンドラスレッドが起動
S1606
割り込みハンドラ本体の実行
S1607
スーパーシグナルスレッド
S1608
多重割り込みか？
YES
NO
スケジューリング機構105呼び出し
S1609
S1610
実行中のハンドラスレッドを中断状態にしてスタック133につむ
S1611
タスク再開機能110の呼び出し
ハードウェア割り込み終了
S1612

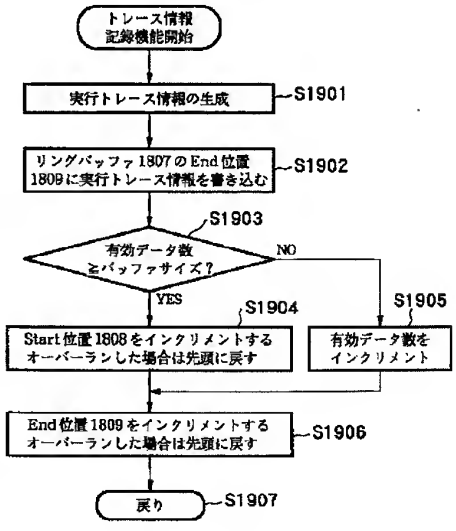
【図１７】


アプリケーション
102
タスク
タスク
タスク
タスク
101
共通部分(ハードウェア非依存部分)
103
システムコールの実体の集合
104
スケジューリング機能
105
機種依存部分
106a
タスクスイッチ制御機能
109a
110a
108a
タスク中断機能
タスク再開機能
111a
132
割り込み管理機能
(スーパーシグナルスレッド)
スタック
レベル4
レベル3
レベル2
レベル1
133
ハンドラスレッド
134
スレッド生成/削除
イベント通知
131
シグナルハンドラ
114a
タスク情報
コンテキスト情報
汎用マシンのOS
(UNIX,Windowsなど)

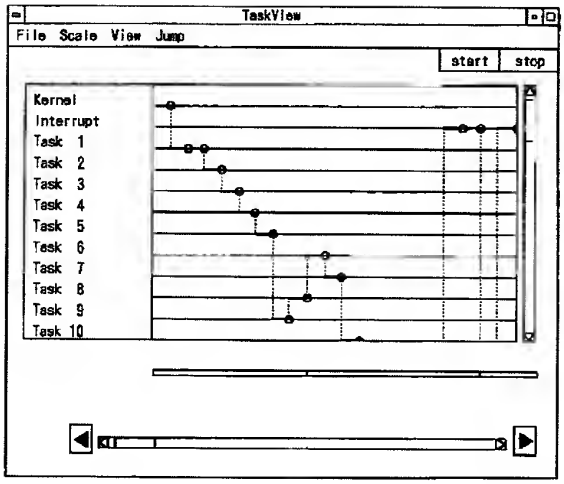
【図１９】


共通部分(機種非依存部分)
103
システムコール・エントリポイント
1801
イベントフラグ
キュー
セマフォ
タスク制御
1802
トレース情報記録機能
1803
デバッグ情報転送機能
メモリ
1804
トレース情報
1809
1807
End
リングバッファ等
Start
有効データ数
1808
機種依存部分
read/write
1805
デバッグ通信モジュール(シリアル・ソケット等)
106
シリアルソケット等
可視化ツール
・タスクスイッチビューア等
1806
WS/PCなど

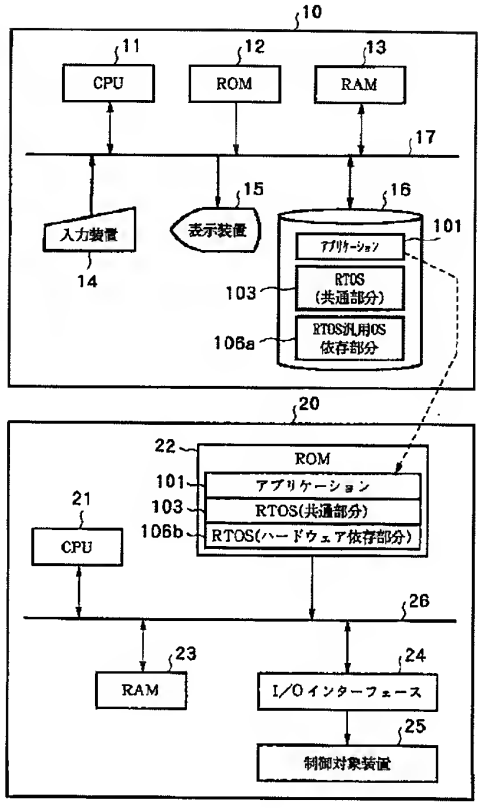
【図２１】


トレース情報記録機能開始
実行トレース情報の生成
S1901
リングバッファ1807のEnd位置1809に実行トレース情報を書き込む
S1902
S1903
有効データ数=バッファサイズ?
NO
YES
S1904
S1905
Start位置1808をインクリメントするオーバーランした場合は先頭に戻す
有効データ数をインクリメント
End位置1809をインクリメントするオーバーランした場合は先頭に戻す
S1906
戻り
S1907

【図２３】



【図２４】



フロントページの続き

(72)発明者　岩渕　洋一
東京都大田区下丸子３丁目30番２号　キヤノン株式会社内

(72)発明者　山田　潤二
東京都大田区下丸子３丁目30番２号　キヤノン株式会社内

Ｆターム(参考)　5B098 GA02 GA04 GA08 GB02 GB09
GB11 GB13 GC03 GC14 JJ07